

***Graph_sampler*: a simple tool for fully Bayesian analyses of DAG-models**

**Sagnik Datta · Ghislaine Gayraud ·
Eric Leclerc · Frederic Y. Bois**

Received: date / Accepted: date

Abstract Bayesian networks (BNs) are widely used graphical models usable to draw statistical inference about Directed acyclic graphs (DAGs). We presented here *Graph_sampler* a fast free C language software for structural inference on BNs. *Graph_sampler* uses a fully Bayesian approach in which the marginal likelihood of the data and prior information about the network structure are considered. This new software can handle both the continuous as well discrete data and based on the data type two different models are formulated. The software also provides a wide variety of structure priors which can be informative or uninformative. We proposed a new and much faster jumping kernel strategy in the Metropolis-Hastings algorithm. The source C code distributed is very compact, fast, uses low memory and disk storage. We performed out several analyses based on different simulated data sets and synthetic as well as real networks to discuss the performance of *Graph_sampler*.

Keywords Bayesian networks · structure learning · posterior distribution · MCMC · Metropolis-Hasting algorithm

Acknowledgment

S.Datta is funded by a Ph.D. studentship for the French Ministry of Research.

Sagnik Datta · Ghislaine Gayraud
Sorbonne Universités, Université de Technologie de Compiègne, BMBI (S. Datta), LMAC
(G. Gayraud) Bât. G.I., CS 60 319 60 203 Compiègne cedex, France
Tel.: +33 3 44 23 47 37 and Fax: +33 3 44 23 44 77
E-mail: sagnik.datta@utc.fr and E-mail: ggayraud@utc.fr

Eric Leclerc
LIMMS/CNRS-IIS (UMI 2820) Institute of Industrial Science, The University of Tokyo,
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
E-mail: eleclerc@iis.u-tokyo.ac.jp

Frederic Y. Bois
INERIS Parc ALATA, BP2 60550, Verneuil en Halatte France

1 Introduction

Representing knowledge with uncertainty and automatic reasoning is often carried out using graphical models (Pearl, 1988; Lauritzen, 1996; Neapolitan, 1990). Judea Pearl and Richard E. Neapolitan were the first to summarize the properties of directed acyclic graphs (DAGs) and established them as a new field of study. In the recent years, formal statistical inference on systems of multiple interacting components is often done using DAGs (Heckerman et al, 1995) or Markov networks (Edwards, 2000). A Bayesian network (BN) or a belief network is a probabilistic model denoted by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which each node or vertex $v \in \mathcal{V}$ represents one of the random variables in set $X = (X_1, X_2, \dots, X_N)$, where N is the number of nodes and each edge $e \in \mathcal{E}$ express the dependence among the variables in X . A BN is always directed and acyclic and is therefore a DAG. Besides static BNs, there are also dynamic BNs (Husmeier, 2003; Friedman et al, 1998), which are actually generalizations of hidden Markov models. In that paradigm, all the random variables of the network are considered to be potentially related to each other over adjacent time points.

DAGs have been used for more than two decades in biomedicine and health-care for handling uncertainty in disease diagnosis, selecting the optimal treatment and predicting treatment outcome (Andreassen et al, 1999). Other applications are found in social network analyses, high dimensional data analyses etc. In computational biology and bioinformatics, BNs have been proposed to model DAGs, for example, for modeling gene regulatory networks, protein structure and gene expression. Many of the available software (Korb and Nicholson, 2010) programs which do BN parameter estimation or structural inference run on commercial platforms. The others which are readily available and free are generally not well maintained and not updated regularly. However there are some widely used packages scripted in R that are free and properly maintained.

In this article we present an efficient C language software *Graph_sampler* for Bayesian inference on the structure of BNs; this latter is based on the well-known Metropolis-Hastings(MH) algorithm that allows to sample DAGs from the posterior distribution. Unlike the existing BN learning software *Graph_sampler* propose a new jumping kernel in the MH algorithm making it more time efficient. The software is easy to use and works well with both discrete and continuous data. For each of these data types we formulated two different models. One of the key feature of this software is that it handles large network structure efficiently. A number of different priors are included in this software to reflect the prior knowledge on the graph structure. This software is quite flexible for using different combination of prior knowledge. We made several performance test on our software with intensive simulation studies regarding all possible choices of models and priors to venture the robustness of *Graph_sampler*. Alongside we considered *structmcmc*, a versatile and easy to use R package (Mukherjee and Speed, 2008) for detailed discussion on the

performance of *Graph_sampler*. Like *Graph_sampler*, this software also aims to learn structure on DAGs via a fully Bayesian approach.

The remainder of the paper is organized as follows: Section 2 briefly discusses the statistical inference for BNs with *Graph_sampler*; Section 3 is about the new jumping kernel, the installation and running of *Graph_sampler*; Section 4 discusses about some of the widely used software and their approach. This section also describes the methodology to check the efficiency of *Graph_sampler* for discrete as well as continuous data set along with the difficulties in the flipping technique. In Section 5 we present our results along with their significance. In Section 6 we summarize all our results in the form of a discussion.

2 Statistical inference for BNs

Recent work on BNs by Mukherjee and Speed (Mukherjee and Speed, 2008), used Markov chain Monte Carlo (MCMC) simulations to infer on network structure from node values. They also considered priors encoding information relative to existence of edges, degree distribution and sparsity structure of the graph. Bois and Gayraud (2015) recently extended informative priors to account for motifs frequencies in order to generate realistic gene regulating graphs (Bois and Gayraud, 2015). For baseline information they proposed to use Bernoulli distributions to model prior knowledge on individual edges.

Variational Bayes (VB) methods have also been proposed as an alternative to full Bayesian inference. VB is an efficient way to deal with intractable integrals arising in a full Bayesian context and can be considered as an extension of the expectation-maximization (EM) algorithm (Beal and Ghahramani, 2003). VB can also help in model selection by providing a lower bound for the data marginal likelihood. However it provides only an approximate analytical solution for the posterior probability of the parameters and latent variables involved in a graphical model. Hence we focus here on a full Bayesian approach.

The following sections describe the prior densities and data likelihood available in *Graph_sampler*.

2.1 Priors on graph structure

It is quite a challenging problem to make inference on graphical model structure even with moderate number of nodes, partly because of the huge number of possible graphs. For N nodes, the number of possible DAGs can be computed recursively as (Robinson, 1973):

$$a_N = \sum_{k=1}^N (-1)^{k-1} \binom{N}{k} 2^{k(N-k)} a_{N-k}$$

where $a_0 = 1$ by convention. So for 2, 3, 4 and 5 nodes network there are 3, 25, 543 and 29281 possible DAGs respectively.

All of those possible graphs are usually not equally plausible a priori and thus certain features may be incorporated as more likely than the others, a priori. To make inference on graph structure, Bois and Gayraud (Bois and Gayraud, 2015) considered three different priors. In the most general case, a set of independent Bernoulli priors (P_B) is used to model the prior knowledge on individual edges. If $p_{i,j}$ is the probability of existence of a directed edge $e_{i,j}$ from node i to node j , and $e_{i,j} \sim B(p_{i,j})$ for all $(i, j) \in \{1, \dots, N\} \times \{1, \dots, N\}$, then the global Bernoulli prior P_B on the graph G is

$$P_B(G) = \prod_{i \neq j=1}^N (p_{i,j})^{e_{i,j}} (1 - p_{i,j})^{1-e_{i,j}}$$

The choice of the value for each hyperparameter $p_{i,j}$ depends on the prior evidence we have on the existence of the given edge from the scientific literature.

The degree $\deg(v)$ of a vertex v is defined as the number of edges involving v . We can also define the degree distribution (P_D) for G as a function $P_d = \text{card}\{v \in \mathcal{V} : \deg(v) = d\}$ for all the vertices having degree d . The prior on degree can be expressed as a power law given by

$$P_d \propto d^{-\gamma}, \quad \text{with} \quad \gamma > 0$$

Thus we can define the prior degree distribution for the graph G as

$$P_D(G) \propto \prod_{i=1}^N \left(\sum_{j=1}^N e_{i,j} \right)^{-\gamma}, \quad \text{with} \quad \sum_{j=1}^N e_{i,j} > 0$$

In addition to P_B and P_D , if we consider a Beta-Binomial prior (P_M) as it is done in Bois and Gayraud (2015) on the occurrence of three-nodes motifs then the total prior on the graph G can be expressed in a product form as:

$$P_T(G) \propto P_B(G) \times P_D(G) \times P_M(G)$$

Alternative to the Bernoulli prior for the presence of edges is the so called concordance prior (P_C) (see Mukherjee and Speed (2008)). The latter required the specification of a prior matrix E with elements $E_{i,j} = 1$ representing a desired edge and -1 representing a non-desired edge. At each iteration the prior is calculated by counting the number of disagreements with the adjacency matrix A with elements $A_{i,j} = 1$ or 0 representing the presence or absence of a directed edge starting from node i and ending in node j of the graph G . The form of the concordance prior is then

$$P_C(G) \propto \exp(-\rho(\sum_{i,j=1}^N |A_{i,j} - E_{i,j}|))$$

where ρ is a positive valued hyperparameter. If P_C is used, then the total prior is:

$$P_T(G) \propto P_C(G) \times P_D(G) \times P_M(G)$$

or, equivalently, if P_B is a flat prior (with $p_{i,j} = 0.5$ for $i \neq j$) to avoid any conflict or double accounting with P_C , the total prior is:

$$P_T(G) \propto P_B(G) \times P_C(G) \times P_D(G) \times P_M(G) \quad (1)$$

2.2 Data likelihood and prior predictive distribution

Our main interest is to uncover the underlying structure of BNs. In any Bayesian network, a parent node has always an influence on its child nodes. Let us denote by $x = (x_1, \dots, x_N)$ the data we have on N nodes, where x_i is a n -dimensional vector; where n is the number of data points per node. Even though the model considered involves many parameters, the posterior distribution of each of these parameters are not of our primary interest. Integrating out the parameters in a Bayesian context leads to the prior predictive distribution (or the joint marginal likelihood of the data): $f(x|G) = \prod_{i=1}^N f(x_i|P_a(x_i))$, where $P_a(x_i)$ is the set of parent values of x_i in graph G (Heckerman et al, 1995) and $f(\cdot)$ is the prior predictive distribution of x_i given its parenthood. For a global parent $P_a(x_i) = \emptyset$ and thus $f(x_i|P_a(x_i))$ reduces to $f(x_i)$.

2.2.1 Continuous data

Gaussian regression model with a Normal-Gamma prior

The general expression for the linear Gaussian regression for a given node x_i is:

$$x_i = M(x_i)\beta + u \quad (2)$$

where $x_i = (x_{1,i}, x_{2,i}, \dots, x_{n,i})$ is a vector of n observations of the dependent variable $x = ((x_{i,j})_{1 \leq i \leq n; 1 \leq j \leq k})$, k is the cardinality of $P_a(x_i)$, $M(x_i)$ is a so-called design matrix of order $(n \times (k+1))$ with the first column as 1's and other columns as $P_a(x_i)$, β is a real valued vector of regression parameters of length k , and u follows a Gaussian distribution $\mathcal{N}(0, \lambda^{-1}I_n)$ distribution with λ being a positive real valued precision and I_n is the identity matrix of size n . The likelihood for this regression model is therefore:

$$L(\lambda, \beta | x_i, P_a(x_i)) = \left(\frac{\lambda}{2\pi}\right)^{n/2} \exp\left(-\frac{\lambda}{2}(x_i - M(x_i)\beta)^t(x_i - M(x_i)\beta)\right)$$

where C^t denotes the transpose of the matrix C .

It is classical to choose conjugate form for the priors on the parameters (i.e. β and λ) involved in the regression model :

$$\begin{aligned} P(\beta, \lambda) &= N_k(\beta | \beta_0, (n_0\lambda)^{-1})Ga(\lambda | \alpha, \omega) \\ &= \frac{(\omega)^\alpha (n_0)^{k/2}}{(2\pi)^{k/2} \Gamma(\alpha)} (\lambda)^{\frac{k}{2} + \alpha - 1} \exp\left(-\frac{\lambda}{2}(\beta - \beta_0)^t n_0 (\beta - \beta_0) - \omega \lambda\right) \end{aligned}$$

where β_0 (real valued vector) and n_0 (matrix of dimension $k \times k$) are hyper-parameters related to β and α, ω , both being real valued positive numbers, are hyper-parameters of λ . In the above equation $\Gamma(\cdot)$ represents the Gamma function.

In that case, the prior predictive distribution is $t_\nu(\mu, \Sigma)$, the n-dimensional multivariate t-distribution with parameters μ, Σ and ν , whose density function is:

$$f(x_i|P_a(x_i)) = \frac{\Gamma(\nu+n)/2}{\Gamma(\nu/2)(\nu\pi)^{n/2}} |\Sigma|^{-1/2} \left[1 + \frac{1}{\nu}(x_i - \mu)^t \Sigma^{-1}(x_i - \mu) \right]^{-\frac{\nu+n}{2}} \quad (3)$$

where, $\mu = [\mu_1, \dots, \mu_n]^t$ is the location parameter, Σ is the scalar matrix of dimension $(k \times k)$ and ν is the degrees of freedom such that

$$\begin{aligned} \mu &= M(x_i)\beta_0 \\ \Sigma^{-1} &= h(M(x_i))\alpha\omega^{-1} \\ h(M(x_i)) &= I_n - M(x_i)[M(x_i)^t M(x_i) + n_0]^{-1} M(x_i)^t \\ \nu &= 2\alpha \end{aligned}$$

Gaussian regression model with a Zellner g-prior

With *Graph_sampler* we can also use the Zellner g-prior (Mukherjee and Speed, 2008; Nott and Green, 2004) for the parameters β and $(\lambda)^{-1}$:

$$\begin{aligned} P(\beta|\lambda^{-1}) &= N_k(0, g\lambda^{-1}[M(x_i)^t M(x_i)]^{-1}), \\ P(\lambda^{-1}) &\propto \lambda \end{aligned}$$

where g is a user defined positive scale factor. The prior predictive distribution of the data is given by:

$$f(x_i|P_a(x_i)) \propto (1+g)^{-(k+1)/2} s^{-n/2} \quad (4)$$

where k is the cardinality of $P_a(x_i)$ and

$$s = x_i' x_i - \frac{g}{1+g} x_i^t M(x_i) [M(x_i)^t M(x_i)]^{-1} M(x_i)^t x_i$$

provided the term $M(x_i)^t M(x_i)$ is invertible. A sufficient condition for $M(x_i)^t M(x_i)$ to be invertible is that $k \leq n$, that is, the number of parents should be less than the number of data points for per node. Thus Zellner g-prior fails to work when number of parents is greater than the number of data points per node.

2.2.2 Discrete data

For discrete data, *Graph_sampler* offers the possibility to use a Multinomial model with a Dirichlet prior on its parameters (see (Heckerman et al, 1995)). For such a prior on the parameters we have a closed form of the prior predictive:

$$f(x|G) = \prod_{i=1}^n \prod_{j=1}^{s_i} \frac{\Gamma(D'_{ij})}{\Gamma(D'_{ij} + D_{ij})} \cdot \prod_{k=1}^{m_i} \frac{\Gamma(D'_{ijk} + D_{ijk})}{\Gamma(D_{ijk})} \quad (5)$$

where D_{ijk} is the number of components of x_i that takes the value k given that $P_a(x_i)$ has configuration j , D'_{ijk} are the Dirichlet hyperparameters, s_i represents the possible number of configurations of the parents of x_i , m_i stands for the number of possible values of components of x_i and D'_{ij} and D_{ij} are given by:

$$D'_{ij} = \sum_{k=1}^{m_i} D'_{ijk} \quad \text{and} \quad D_{ij} = \sum_{k=1}^{m_i} D_{ijk}.$$

3 Graph_sampler: sampling and installation

3.1 Efficient sampling: fast jumping kernel

Graph_sampler can efficiently generate random samples for general directed graphs (Bois and Gayraud, 2015), but we focus here on the sampling of BNs from a posterior distribution conditioned by data (observed node values). We use an adjacency matrix representation for the graph and store only the eventual difference between adjacency matrix as it is a fast and efficient storage method. A Metropolis-Hasting sampler (Robert and Casella, 2004) is used to sample random graphs according to the prescribed posterior probability distribution. The algorithm of the simplified jumping kernel is as follows for the t -th iteration:

We denote the current graph by G and its adjacency matrix by A^{G^t} . The proposal graph is denoted by G' and its adjacency matrix by $A^{G'}$ in our algorithm.

Algorithm

Step 1: Select $A_{i,j}^{G^t}$ while scanning A^{G^t}

Step 2:

- (a): Sample $z_{i,j} \sim \text{Bernoulli}(p_{i,j})$ where $p_{i,j}$ is the Bernoulli prior for edge $e_{i,j}$
- (b): (i): *Adding an edge*

$$\text{if } z_{i,j} = 1, \text{ then } A_{i,j}^{G'} = \begin{cases} A_{i,j}^{G^t} & \text{if } A_{i,j}^{G^t} = 1 \\ 1 & \text{o.w. and provided } G' \text{ is still a DAG,} \\ & \text{o.w. go back to Step 1} \end{cases}$$

(i): *Deleting an edge*

$$\text{if } z_{i,j} = 0, \text{ then } A_{i,j}^{G'} = \begin{cases} A_{i,j}^{G^t} & \text{if } A_{i,j}^{G^t} = 0 \\ 0 & \text{if } A_{i,j}^{G^t} = 1 \end{cases}$$

Step 3: Calculate the acceptance ratio. We accept G' with probability

$$\delta = \min(1, \frac{f(x, G') P_T(G') P(A_{i,j}^{G^t} | A_{i,j}^{G'})}{f(x, G^t) P_T(G^t) P(A_{i,j}^{G'} | A_{i,j}^{G^t})})$$

Note that due to Step 2, the acceptance ratio can be rewritten as follows:

$$\delta = \min(1, \frac{f(x|G') P_T(G') P_B(G^t)}{f(x|G^t) P_T(G^t) P_B(G')})$$

where $f(x|G) = \prod_{i=1}^N f(x_i | P_a(x_i))$ is the prior predictive given by Eq (3, 4 and 5) and P_T is the total prior on the graph structure. Clearly this simplifies since P_B is a part of P_T (see Eq(1)).

Step 4: Choose G^{t+1} as follows: $G^{t+1} = \begin{cases} G' & \text{with probability } \delta \\ G^t & \text{with probability } 1-\delta \end{cases}$

The procedure is repeated until convergence in probability is attained. Gelman and Rubin's (GR) \hat{R} criterion (Gelman and Rubin, 1992) is used on each element of the graph's adjacency matrix to check the convergence of several simulation chains. The advantage of using the GR criterion for the convergence is that, we do not have to save the whole chains from the start. We can check the convergence using the final posterior edge probability matrix obtained after the specified number of iterations neglecting the burning runs. For this criterion, we consider the three edge probability matrices and calculate the within-chain and the between-chain variance. Then the estimated variance of the parameter is calculated as a weighted sum of the within-chain and between-chain variance. Based on the potential scale reduction factor we infer on the convergence of the three chains. For BNs we need to ensure that the proposed graphs are DAGs. This is done with a fast topological sorting algorithm (similar to that of (Pearce and Kelly, 2006)) operating on a list index of the nodes.

3.2 Graph_sampler installation

Graph_sampler is an easily available free software that can be redistributed or modified under the terms of the GNU General Public License as published by the Free Software Foundation. It is an inference as well as simulation tool for DAGs and can simulate random graphs for general directed graphs as well as for DAGs. In the case of BNs, we infer about their probable structure through the joint use of priors and data about node values.

Graph_sampler is written in ANSI-standard C language and can be compiled in any system having a ANSI C compliant compiler. The GNU gcc compiler (freeware) is highly recommended and the automated compilation script (called Makefile) can be successfully used if the standard 'make' command is available. In order to modify the input file parser, the 'lex' and 'yacc' are highly recommended. The full software along with the manual can be downloaded from:

<https://sites.google.com/site/utccchairmbsptp/software>

Once downloaded, the software should be decompressed using 'gunzip' and 'tar' commands. Other archiving tools can also be used. *Graph_sampler* can be compiled using the 'make' command. On successful compilation of *Graph_sampler*, it is ready for running. In order to run *Graph_sampler*, an input file specifying the simulation parameters should be provided. In Unix the command-line syntax to run that executable is:

"*graph_sampler* [*input-file* [*output-prefix*]]"

where the brackets indicate optional arguments. If no input file and/or output prefix are not specified, the program uses the defaults. The default input file is *script.txt* and the output files created depends on the parameters specified in the input file. Default output file names are *best_graph.out*, *graph_samples.out*, *degree_count.out*, *motifs_count.out*, *edge_p.out* and *results_mcmc.bin*.

A *Graph_sampler* input file is a text (ASCII) file that obeys relatively simple syntax (see the manual). Values of all the predefined variables in the input file should be properly defined. Description and range of each variable is illustrated in the manual. In case of improper assignment of values, *Graph_sampler* post error messages during runtime.

4 Efficiency analyses of *Graph_sampler*

In order to evaluate the efficiency and accuracy of *Graph_sampler*, we performed several experimental runs based on different network structure, size and data type. We even varied the underlying model based on the data type to have a clear idea about the efficiency of the software. There are other very well known packages in R like the *deal* and *bnlearn*. The package *deal* (Boettcher and Dethlefsen, 2003) is scripted in R language and uses BNs to analyse the data which can be discrete and/or continuous types; for the network parameters, suitable priors can be constructed and parameter estimation is possible using successive updating. This package is useful for structure learning of the network and uses the heuristic greedy search algorithm. The scoring function is based on maximising the Bayes factor. At each step of the greedy search algorithm, we either add, delete or reverse an edge and calculate the Bayes factor for all the possible graphs. The proposal graph with the maximum Bayes factor is selected to update the current graph. The package *bnlearn* (Scutari, 2010) is also scripted in R to do structural inference on BNs. This package is efficient to work with both discrete as well as continuous data. For the BN structure

learning, various constraint-based algorithms like the *Grow-shrink*, *Incremental association* and *Max-min parents and children* algorithms are implemented. *Hill-Climbing* greedy search algorithm is the only score-based algorithm implemented in this package. The sampling algorithm consists of first recovering the skeleton of the desired graph and then evaluating the directionality of the edges. Both of these packages sample the best graph that maximizes the Bayes factor or the posterior odds from amongst all possible proposal graphs. This is quite different from the sampling scheme that we proposed. Our interest is to sample graphs based on the specified posterior distribution. Another efficient Matlab package for BN analysis is *bnt* (Murphy, 2007). This tool does inference on parameters as well as on network structure. This software is suitable for both decision networks and dynamic Bayesian networks. However this tool lacks the GUI. Inference is carried out using various algorithms like the Junction tree, variable elimination and Pearl’s polytree. *bnt* has various options for parameter learning as well as for structure learning. It is very clearly documented, free and is very object-oriented. Besides these well known packages, we also have *structmcmc* (Mukherjee and Speed, 2008) which is an efficient software coded in R for BN structure learning. We selected *structmcmc* because of its similarities with *Graph.sampler* regarding the model formulation and the Metropolis-Hasting algorithm. Unlike *Graph.sampler*, the sampling technique in *structmcmc* also allows flipping of edges between two selected nodes. This package also propose to sample graphs based on the posterior distribution. Thus we selected *structmcmc* as a baseline software to discuss in details the advantages and limitations of *Graph.sampler*. *structmcmc* proposes a Zellner g-prior for continuous datasets and a Multivariate Dirichlet prior for discrete datasets. The Normal Gamma model is not available to *structmcmc* users.

In all the cases we used a null matrix as the initial adjacency matrix. Results showed that they are robust with respect to the initial adjacency matrix. For the prior on edges, we used the concordance prior (Mukherjee and Speed, 2008) (P_C) with $\rho = 1$ since *structmcmc* does not provide a Bernoulli prior. The prior on the loop motifs P_M was not used and therefore set to 1. We also used the degree prior (P_D) to check the increase in efficiency of *Graph.sampler*. As an alternative to (P_C), we also used both informative as well as uninformative (P_B) prior as a structure prior to check the efficiency of the software. We followed the simulation procedure described in (Mukherjee and Speed, 2008) to generate discrete datasets. For the continuous case, we generate data as described in Equation (2).

To start with we considered a real life biological network specifically the EGFR system. For this actual network we simulated a discrete dataset. This real life network consisted of only 14 nodes. In order to check efficiency of *Graph.sampler* for larger networks, we simulated networks of 5 to 120 nodes, with 100 data points for each node. Figure 1 represents the network with 120 nodes. It is clearly a descending tree network. We used three different seeds to run three chains for each software program. We saved the three chains separately and calculated Gelman’s \hat{R} convergence diagnostic at each iteration.

The first iteration for which \hat{R} attained at most 1.05 for all edges of the graph was considered as the minimum number of iterations required for convergence. *Graph_sampler* was compiled with gcc version 4.2.1 (Apple Inc. build 5666) while *structmcmc* was run with R 3.0.2 (R Core Team, 2013). We performed a time and convergence comparison between both the software. In order to check the performance and efficiency of both the software, we use R language script to plot the heat map and the accuracy curve. The heat map is used to summarize for all edges the edge posterior distribution through its means. We used the R language package *lattice* to plot the heat maps. The accuracy curve is given by $accuracy = (true\ positive\ edges + true\ negative\ edges) / total\ number\ of\ edges$ and is a function of the probability threshold above which an edge is declared present. We used *SDMTools*, to plot the accuracy curves.

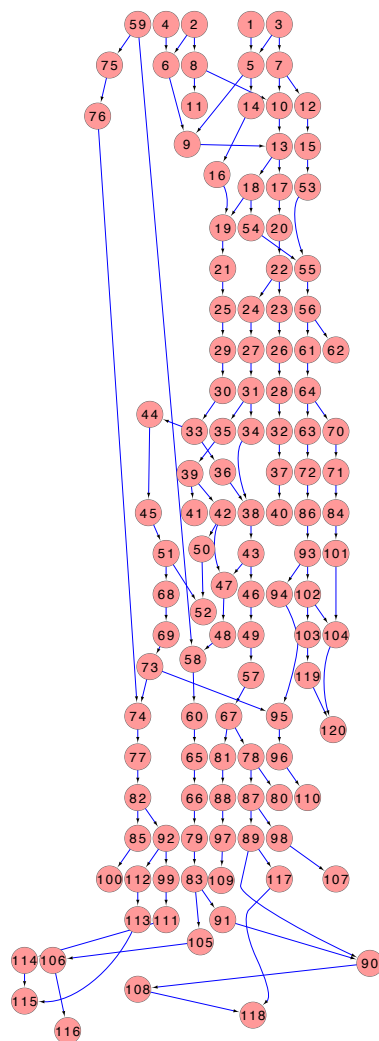


Fig. 1: Hierarchical representation of the 120 nodes network used to generate our simulated data. All the other smaller networks were subsets from this network

5 Results

5.1 Dealing with Discrete data

5.1.1 An actual biological network

To study the practical efficiency of *Graph_sampler* for a true biological network, we considered the same network as studied by S. Mukherjee (Mukherjee and Speed, 2008). In their work they considered a biological network known as the epidermal growth factor receptor (EGFR) system. Figure 2 gives a pictorial representation of the EGFR system. This biological network involves 14 proteins each of which is a ligand, a receptor or a cytosolic protein. Data for this study was synthesized based on the network and following the model as used by S. Mukherjee (Mukherjee and Speed, 2008). Each of the 14 proteins are considered to be a random variable with binary values $\{0, 1\}$. Depending on the parents, the conditional distributions are defined as Bernoulli with success parameter p . The global parents are sampled with $p = 0.5$. For the child nodes, we take $p = 0.8$ if one of its parents take the value 1 and $p = 0.2$ otherwise. For each node we simulate 200 data points.

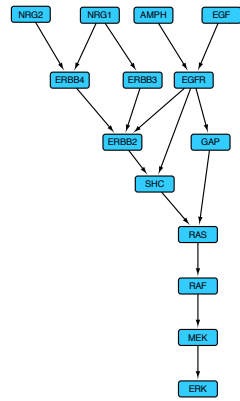


Fig. 2: Graph of the EGFR system

To make our experimental runs coherent with *structmcmc*, we defined the prior matrix on the graph structure based on the concordance prior proposed in (Mukherjee and Speed, 2008). As the model proposed was Multinomial model, we defined a Dirichlet prior for the parameters involved. In case of *Graph_sampler* we strengthen the prior on the structure of the network by considering a degree prior with $\gamma = 3$.

Figure 3 represent our comparative study in the form of heat maps and accuracy curve. We observe that for a small network like the EGFR system there

is no significant difference in accuracy between *Graph_sampler* and *structmcmc* for low threshold values but for higher threshold values, *structmcmc* has an advantage. Observing the heat maps we find that even though *Graph_sampler* produces lower edge probability matrix but does not give rise to false negative edges. On the other hand *structmcmc* generates high edge probability matrix but leads to a higher number of false negative edges that were not present in the prior network. Thus we can take this as a trade off where we have to be careful while working with both of these software and decide accordingly. If we focus on time efficiency of the two software, we observe that *Graph_sampler* is almost 200 times faster than *structmcmc*. Thus taking both the criteria together, we observe that *Graph_sampler* has its own advantages and limitations like *structmcmc*.

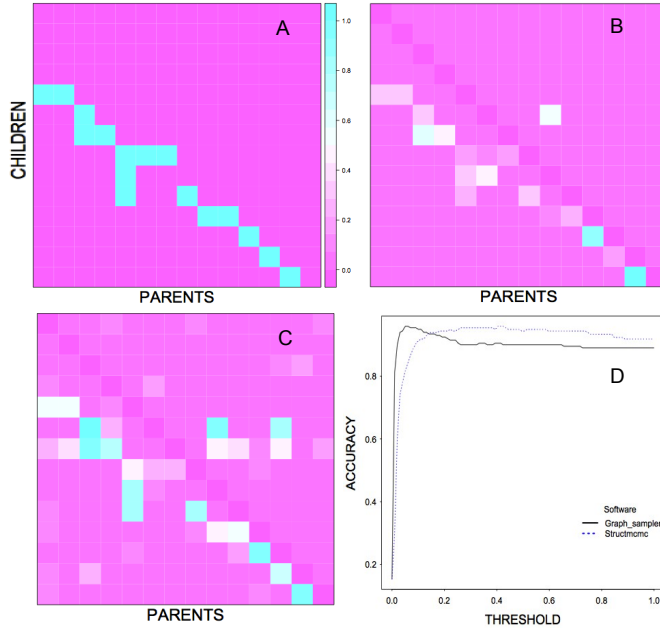


Fig. 3: Heat map of the real network with 14 nodes (A), edge posterior heat maps of *Graph_sampler* (B) *structmcmc* (C) with discrete data. The x-axis represents the parent nodes and the y-axis the corresponding children. Accuracy curve of the two software (D).

An alternative other than the Dirichlet prior on the parameter could be the Zellner g-prior with g-prior equals 1. Even though the Zellner g-prior should be used for continuous data, we did use it for the discrete data to check its efficiency. For a binary data set as described above, we can easily fit a linear regression model and thus fall back to the continuous scenario making the use

of Zellner prior valid. We still used the priors P_C and P_D . Figure 4 represents the heat map and the accuracy curve for the two software. Comparing the heat map of figure 3(B) and figure 4(A), we observe that there is an improvement in the posterior edge probability matrix obtained from *Graph_sampler*. There is also a reduction in the number of false negative edges and this is true even for *structmcmc*. As far as accuracy is concerned, both the software are quite efficient and there was no significant difference in accuracy between the two. However considering the time scale, *Graph_sampler* is again 100 times after than *structmcmc* and thus have a slight advantage.

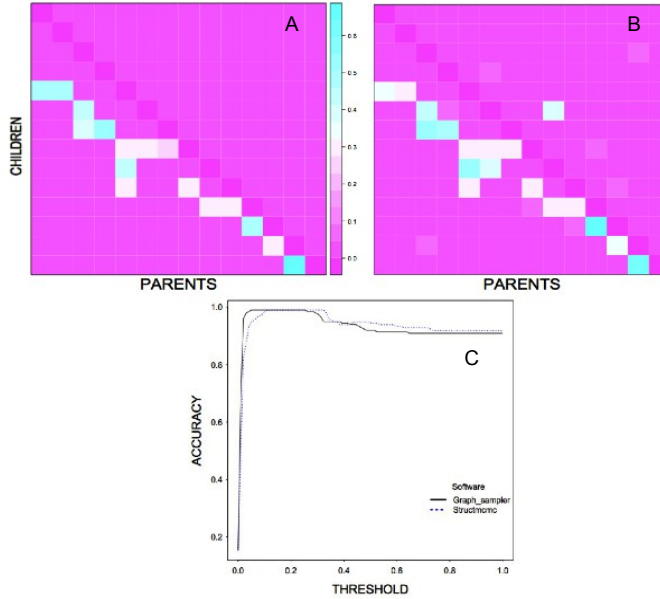


Fig. 4: edge posterior heat maps of *Graph_sampler* (A), *structmcmc* (B) with Zellner g-prior, here the x-axis represents the parents and y-axis the corresponding children. Accuracy curve for the two software (C)

5.1.2 Simulated networks

We studied the performance of *Graph_sampler* for discrete datasets, using the Multinomial model. Figure 5 gives a graphical summary of our timing results. Because of memory problems we could not achieve convergence with *structmcmc* for networks of more than 60 nodes. Similarly for 120 nodes, *Graph_sampler* did not converge with a billion iterations. One of the reason behind this could be that with larger network size we have to increase the dataset. *Graph_sampler* was about 100 times faster than the *structmcmc* for

the same number of iterations. *Graph_sampler* running time was also less influenced by the network size (i.e. it increased by a factor 4.06 when going from 5 to 60 nodes with *Graph_sampler* and that factor being 16.5 for *structmcmc*). With 30 nodes, *Graph_sampler* took about 2×10^7 iterations (275 seconds) to converge while for *structmcmc* it was 10^6 iterations (3848 seconds). Thus *structmcmc* required 10 to 100 times less iterations to reach convergence but was about 14 times slower than *Graph_sampler*. We compared the edge probability matrices from both the software by the accuracy curve to check whether they converge to the true graph.

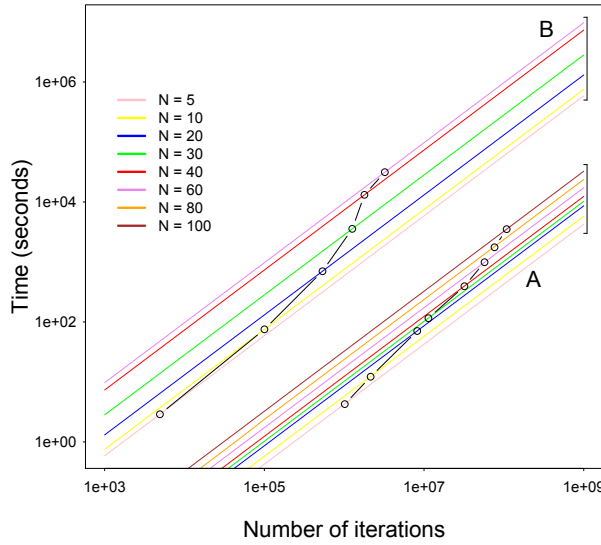


Fig. 5: Time and convergence comparison of *Graph_sampler* (A) and *structmcmc* (B) performance for various network size (N) with discrete data. The x-axis represents the number of iterations performed and the y-axis the time taken on an Apple mac book 2.53 GHz Intel Core 2 Duo processor. The black lines with circles give the minimum number of iteration required to reach convergence.

Figure 6 panels (A - C) represent the posterior edge probabilities in the form of heat maps. Figure 6 panel (D) shows the accuracy of the software in retrieving actual edges as a function of the probability threshold above which an edge is declared present. It was observed that with small threshold values *structmcmc* had higher accuracy than *Graph_sampler*. We observe that with the single use of concordance prior the accuracy of *Graph_sampler* in retrieving edges correctly is low for small threshold values. However with a

threshold value above 0.5, both the software had almost the same accuracy. Altogether, *Graph_sampler* reaches convergence faster in time and has equal accuracy as *structmcmc* for threshold values above 0.5.

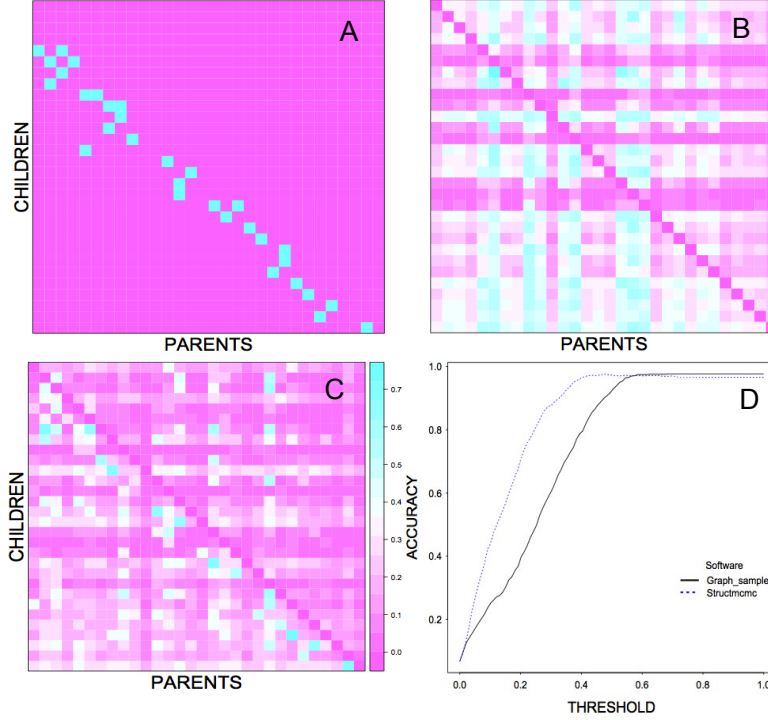


Fig. 6: Heat map of the true network with 30 nodes (A), edge posterior heat maps of *Graph_sampler* (B) *structmcmc* (C) with discrete data. The x-axis represents the parent nodes and the y-axis the corresponding children. Accuracy curve of the two software (D).

The accuracy of *Graph_sampler* can be improved with the introduction of the degree prior (P_D) with $\gamma = 2$. As our network is a descending tree network, the inclusion of the degree prior is very beneficial. Figure 7(A) represents the posterior probabilities of the edges and resembles more like the true graph. It was observed that with the prior P_D , the accuracy of *Graph_sampler* improved significantly (Figure 7(B)) and even with very low threshold values, the accuracy was almost equal to 0.9.

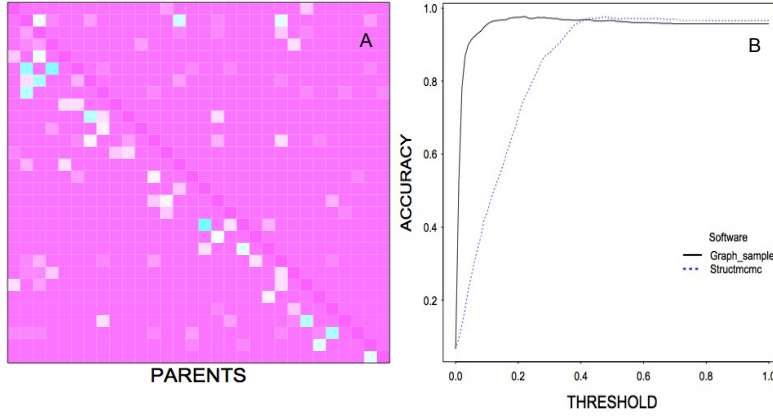


Fig. 7: Heat map of the posterior edge probabilities obtained from *Graph_sampler* with discrete data with concordance and degree priors (A) and the improvement in the accuracy curve due to the use of degree prior in *Graph_sampler* (B).

5.2 Dealing with Continuous data

For continuous data, *structmcmc* uses a Zellner g-prior on regression parameters. With *Graph_sampler* either a normal-gamma prior or a Zellner g-prior can be used. The two models are compared here.

5.2.1 Normal-gamma model

For the continuous data set, convergence with *Graph_sampler* was achieved with almost 10 times less iteration number compared to *structmcmc*. It was observed that with networks having 60 nodes, *structmcmc* faced problems in convergence. Figure 8 represents our study in a graphical way. Regarding the time taken for iterations, *Graph_sampler* was almost 10 times faster than *structmcmc*. For networks with more than 20 nodes, time efficiency of *structmcmc* decreases sharply. The very narrow band width (A) reveals that for *Graph_sampler* the increase in network size does not have much influence on time. Figure 8 represents our study in a graphical way.

We also studied the posterior edge probabilities obtained from each of the software to draw inference on their efficiency to retrieve the true graph structure. We plotted the posterior edge probabilities in the form of heat maps to understand sampling scheme prescribed in *Graph_sampler*. Figure 9 represents the three heat maps for a network with 30 nodes. We observe that *Graph_sampler* perform well in retrieving edges present higher up in a tree network. However as we go down the tree structure, the efficiency of *Graph_sampler* decreases. As our model used simulated data, so there is an underlying correlation in the dataset. For this reason as we go down the network

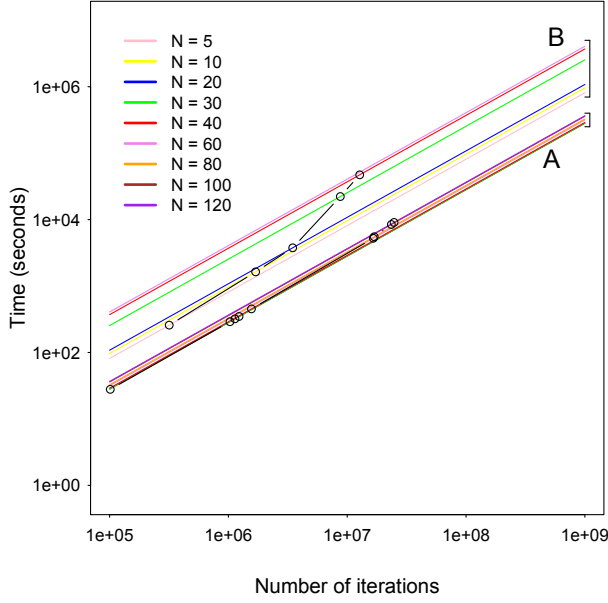


Fig. 8: Time and convergence comparison of *Graph_sampler*(A) with the *structmcmc*(B) for varying network size with continuous data set. The *Graph_sampler* uses a normal gamma prior while *structmcmc* uses a Zellner g-prior

structure *Graph_sampler* samples many new edges. Interestingly the directionality of the network structure is maintained. On the other hand *structmcmc* is efficient as we move down the network structure. However *structmcmc* fumble with the directionality of the edges. We plotted the accuracy curve for the two software. Figure 9(D) represents that the accuracy of *structmcmc* is slightly higher than that of *Graph_sampler* for smaller threshold values. However this difference is not much and for higher threshold values, both the software have almost equal accuracy.

5.2.2 Strengthening the prior model by considering the hierarchy of the structure and defining a degree prior

The degree prior (P_D) with $\gamma = 2$ was introduced to check the improvement in the accuracy of *Graph_sampler*. Figure 10(A) represents the posterior probabilities of the edges and resembles more like the graph in Figure 9(B). It was observed that with the prior P_D , there was no significant improvement in the accuracy of *Graph_sampler* (Figure 10(B)).

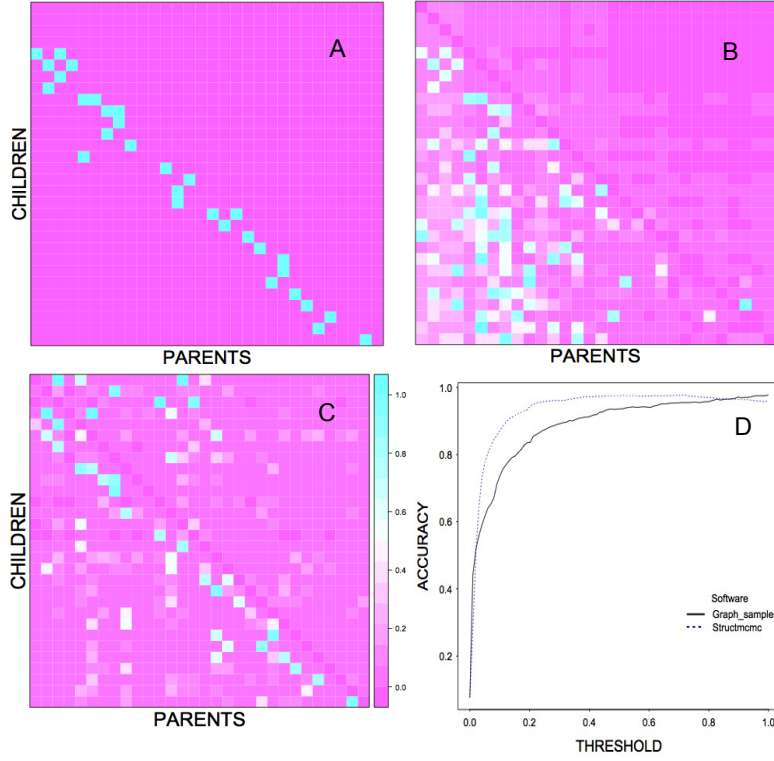


Fig. 9: Heat map of the true network with 30 nodes (A), edge posterior heat maps of *Graph_sampler* (B) *structmcmc* (C) with continuous data, the x-axis represents the parents and y-axis the corresponding children. Accuracy curve for the two software (D).

5.2.3 An alternative: Zellner g-prior model

For continuous datasets, we can also use the Zellner g-prior as used in *structmcmc*. Unlike *structmcmc*, we run *Graph_sampler* for various g-prior values. We started with a g-prior of 1 for all the networks considered in our study. We checked the time required and the convergence point. Later we performed similar runs for different g-prior with values 5, 10, 50 and 100. In each case we observed the convergence rate and the posterior edge probabilities.

With a g-prior value of 1 or 5, convergence was achieved for all the networks. As we increased the value of g-prior to 10, runs with networks having 5 and 10 nodes converged with the maximum value of \hat{R} being 1.12. This was not the case for networks with 20, 30, 40 and 60 nodes as they converged ($\hat{R} = 1.05$ approx). However increasing the g-prior value to 50 and 100, convergence was not achieved for any of the networks. Thus in such a situation, *Graph_sampler* and *structmcmc* differs. For smaller network sizes, *Graph_sampler* performed

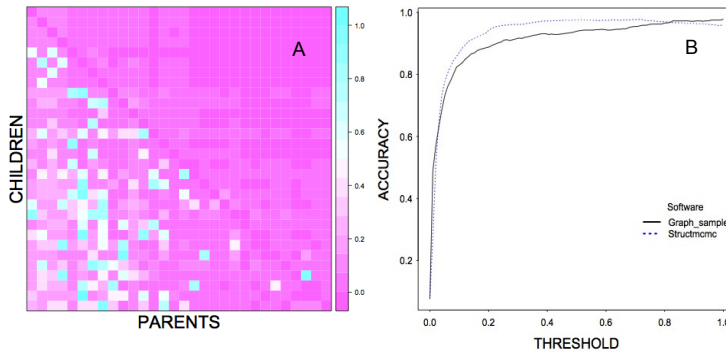


Fig. 10: Heat map of the posterior edge probabilities obtained from *Graph_sampler* with Normal-gamma model with concordance and degree priors (A). Accuracy curve with the use of degree prior in *Graph_sampler* (B).

well with g-prior value equal to 1 or 5. With bigger networks sizes having 100 data points for each nodes, the g-prior value can range from 1 to 40. *Graph_sampler* failed to converge when the g-prior value was equal to the number of data points. On the other hand, *structmcmc* performed well with higher g-prior values and was most efficient when the g-prior value was equal to the number of data points.

In order to understand the reason behind such a difference in convergence rate between the two software, we discuss the flipping technique used in *Graph_sampler* along with its advantages and disadvantages.

5.2.4 Problem with flips

The primary advantage of the reduced jump kernel (only adding or deleting one edge at a time) used in *Graph_sampler* is that it is faster than the jump kernel allowing flips. Since the choice of pairs of nodes is systematic, there is no need to check the neighbourhood cardinality (Husmeier, 2003) as done in *structmcmc*.

This jump kernel has some drawbacks also. Consider a network with 5 nodes where node 4 is a parent of node 5. In the MCMC simulations, at a particular step, we propose to add an edge from node 5 to node 4. As the log posterior for the proposed network is quite high (if 4 conditions 5, the two are correlated and 5 conditioning 4 has high probability), we accept such a proposal. According to our flipping technique, in order to retrieve the true edge (from 4 to 5), we need to first delete the edge from 5 to 4, leaving them independent. However a network with 4 and 5 independent has low log posterior probability and we rarely accept such a move.

Figure 11 is a dot plot where the blue dots and the red circles appearing in pairs represents the difference in log probability for a network when passing from an edge (4 to 5) to an edge (5 to 4) respectively using the jump kernel

specified in *Graph_sampler*. For a pair A, a move from the red dot (log probability -675) to a blue dot (log probability -676), the log probability has to pass through -720 (4 and 5 independent) making such a move impossible. So in the most likely regions of G the flip is very unlikely. For the pair B, a move from the red dot (log probability of -698) to a state of independence (log probability of -685) is easy, but the next move to a blue dot (log probability of -700) is not easy. This is also an unlikely region where some flips are possible. The flip for the pair C is unlikely to occur as the log probability has to pass by -687 while going from -678 to -681. Flips would occur easily when they are close to the diagonal.

The difficulty with the jump kernel can be due to the large data set for which the posterior mass favours fewer graphs. Under such a situation, the standard MCMC scheme faces difficulty in moving between graphs, or finding the high-scoring graphs. In such a case parallel tempering is a proficient option to speed up the MCMC-based convergence of network inference. This tempering approach is generally referred to as Model Composition by Metropolis-Coupled Markov Chain Monte Carlo (MC⁴) (Barker et al, 2010). The parallel tempering involved in this MC⁴ (beyond the scope of this paper) approach allows proper mixing of the Markov chain and helps to escape the local maxima.

5.2.5 Sensitivity to the prior on graph structure

To check the sensitivity of the normal-gamma model with respect to informative and non informative priors, we considered a network with 40 nodes having 100 data points for each node and defined only the P_B priors on the edges. We first considered an informative prior on edges with each desired edge having a prior probability of 0.9 and 0.1 for others except for autoloops for which probability was 0. We define a less informative prior with 0.8 and 0.2 and carry out our experimental run. We repeated the process and finally defined a flat non informative prior of 0.5 for all the edges.

For each run with different prior probabilities, convergence was obtained in *Graph_sampler* and then retrieved the posterior edge probabilities as it conveyed the information regarding the sensitivity of *Graph_sampler* in selecting the desired graph out of all the equivalent graphs.

With a strong informative prior of 0.9 and 0.1, *Graph_sampler* converged and was able to retrieve all the desired edges with a high probability. The posterior probabilities of some undesired edges were also high. This is mainly observed as we move down the network due to the presence of partial correlation between the nodes higher up in the network and those at the bottom. As we use less informative priors, this behaviour becomes more prominent and the efficiency of *Graph_sampler* decreases (Figure 12). With a flat prior of 0.5 for all the edges, the efficiency of *Graph_sampler* is the least. Figure 1 and the heat map of Figure 12(A) show that the true network is a descending tree (the upper triangular matrix of the heat map has zero edge probability). For the informative prior, we observe that the normal-gamma model works well

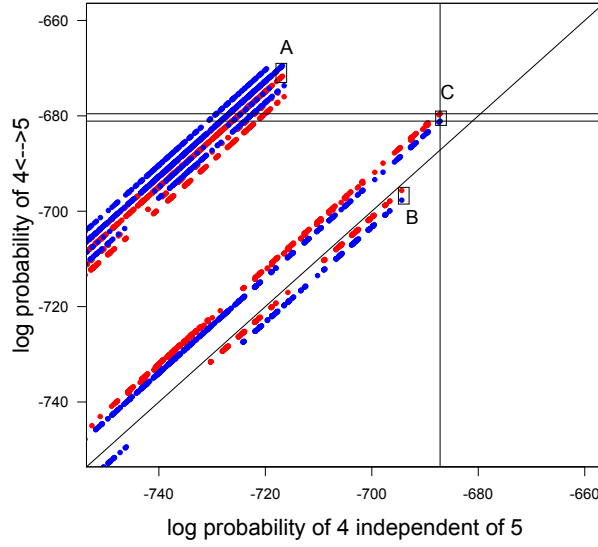


Fig. 11: The flips for a network with 5 nodes along with the log posterior probability after each flip. The blue dots represent the presence of the edge from 4 to 5 and the red dots for the edge from 5 to 4 in a 5 node network. The vertical and the horizontal lines in the graph are kept for easy reference of the marked pairs with both the axis.

for the upper part of the tree network. However as we descend down the tree, the sensitivity of the model decreases as more children are involved. One way to increase the performance of the model can be by increasing the number of data points for each node present in the lower part of the network. We plotted the accuracy curve to depict the efficiency of *Graph_sampler* for the various informative priors.

Figure 12(D) represents the accuracy curve of *Graph_sampler*. We observed that *Graph_sampler* was very versatile with the type of prior information provided. With very strong prior the accuracy was almost equal to 0.9 for threshold values above 0.2. This stated that *Graph_sampler* was efficient enough to retrieve the true edges with higher posterior probabilities and allot low probability (less than the threshold) to false edges. For noninformative priors *Graph_sampler* had an accuracy of 0.65 for threshold below 0.3. The accuracy increased to 0.8 and above with the increase in threshold from 0.5 to 1.0.

We checked the sensitivity of *Graph_sampler* for degree prior (P_D) with a noninformative Bernoulli prior on edges. We observed that there was not much improvement in the accuracy of *Graph_sampler* in this case (Figure 13).

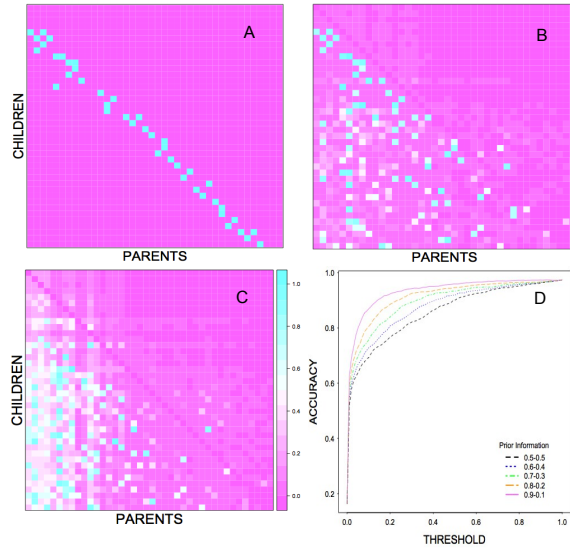


Fig. 12: Heat map of the true (desired) network with 40 nodes (A), heat map of posterior edge probabilities obtained from *Graph_sampler* with an informative prior of 0.9 for desired edges (B) and with a noninformative prior of 0.5 (C) with a Normal gamma likelihood; the x-axis represents the parents and y-axis the corresponding children. Accuracy curve for the various informative priors (D)

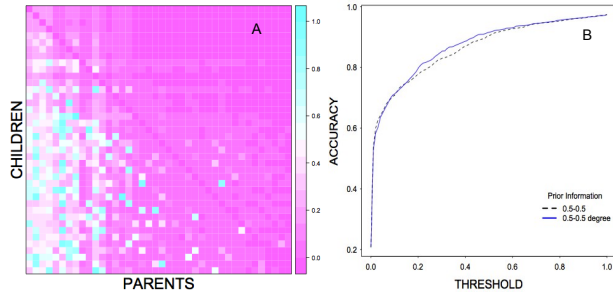


Fig. 13: Heat map of the posterior edge probabilities obtained from *Graph_sampler* with Normal-gamma model with a noninformative Bernoulli prior and degree priors (A). Accuracy curve due to the use of degree prior in *Graph_sampler* (B).

6 Discussion

We observed that like other widely used software (*bnlearn*, *deal*), *Graph_sampler* works well with both discrete as well as continuous data. For continuous data,

we considered a regression model and we fitted a multinomial model for the discrete data. These two model representations are commonly used when dealing with BNs. For continuous data, *Graph_sampler* allow the user to use either the Normal-gamma prior or the Zellner g-prior for the parameters involved in the regression model. Our results showed that *Graph_sampler* performs better with the Normal-gamma prior than the Zellner g-prior. Another drawback of using a Zellner g-prior is that while using this prior the number of parents should be less than the number of data points per node which can be unrealistic while dealing with real observations. For the multinomial model we defined a Dirichlet prior for the parameters. This software provides a number of different structure priors which are mutually independent and can be used in varied combination to improve the results. In particular we considered the concordance prior (P_C), the Bernoulli prior (P_B), the degree prior (P_D) and the motif prior (P_M). Even though we did not utilize the motif prior in this work, but one can use this prior according to the needs. Taking into consideration the data type and the structure prior we are able to perform full Bayesian analyses to sample graphs from the specified posterior distribution. We observed that while dealing with large networks the choice of priors and their combination is quite essential for retrieving the true edges because of the huge number of possible DAGs. It means that larger networks generally require strong informative priors on the structure of the network. Hence for specific network structures (like, tree structure), we have to think of incorporating a more appropriate structure prior. For a large tree structure we observed that the efficiency of *Graph_sampler* in retrieving the true edges decreases as we move down the tree. An alternative way to solve this problem could be to use larger data sets for nodes present way down in the tree structure.

We find that *Graph_sampler* is very efficient with respect to time. Both for continuous as well as discrete data, we observed that *Graph_sampler* was atleast 10 times faster than *structmcmc*. One of the primary reason behind *Graph_sampler* being time efficient is due to its fast jumping kernel. The systematic scanning of the edges and proposing to add or delete an edge at each iteration makes this kernel very efficient with respect to time. Moreover this jumping kernel requires less memory for storage. However as explained in Section 5, this jumping kernel has some limitations. The kernel faces difficulty with large data sets for which the posterior mass favours fewer graphs. The standard MCMC scheme faces problem with the local maxima and thus fails to search for the high-scoring graphs. A proficient option would be to use parallel tempering (beyond the scope of this paper) to speed up the MCMC based convergence. Secondly, *Graph_sampler* is scripted in C language. C language being a compiled language is much more faster than R which is an interpreted language. Our results showed that for large networks i.e. networks with more than 100 nodes, *Graph_sampler* was efficient in convergence. To resume, we observed that *Graph_sampler* is a flexible software which is efficient with respect to time and convergence even for large networks.

We observed that *Graph_sampler* could be a good software apart from the widely used R packages to perform Bayesian analyses of DAGs models. Besides

structmcmc, *bnlearn* and *deal* which are scripted in R, there are several other very efficient BN software that are capable of performing Bayesian inference on parameter estimation and/or on network structure. Some of these software are free while others run on commercial platforms. Each software run on different platforms and follow specific sampling scheme. For interested readers, Korb and Nicholson (2010) reviewed some of the software available at the time that do inference on network structure.

References

- Andreassen S, Riekehr C, Kristensen B, Schonheyder H, Leibovici L (1999) Using probabilistic and decision-theoretic methods in treatment and prognosis modeling. *Artificial Intelligence in Medicine* 15:121–134
- Barker D, Hill S, Mukherjee S (2010) Mc4: A tempering algorithm for large-sample network inference. *Pattern Recognition in Bioinformatics* 6282:431–442
- Beal MJ, Ghahramani Z (2003) The Variational Bayesian EM Algorithm for Incomplete Data: with Application to Scoring Graphical Model Structures, Oxford University Press, pp 453–464
- Boettcher S, Dethlefsen C (2003) deal: A package for learning bayesian networks. *Journal of Statistical Software* 8:1–40
- Bois F, Gayraud G (2015) Probabilistic generation of random networks taking into account information on motifs occurrence. *Journal of Computational Biology* 22(1):25–36
- Edwards D (2000) Introduction to graphical modelling, 2nd edn. Springer, New york, USA
- Friedman N, Murphy K, Russell S (1998) Learning the structure of dynamic probabilistic networks. In: *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pp 139–147
- Gelman A, Rubin DB (1992) Inference from iterative simulation using multiple sequences. *Stat Sci* 7:457–511
- Heckerman D, Geiger D, Chickering D (1995) Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning* 20:197–243
- Husmeier D (2003) Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic bayesian networks. *Bioinformatics* 19(17):2271–2282
- Korb K, Nicholson A (2010) *Bayesian Artificial Intelligence*. CRC Press
- Lauritzen S (1996) *Graphical models*. Oxford University Press
- Mukherjee S, Speed P (2008) Network inference using informative priors. *Proc Natl Acad Sci USA* 105(38):14,313–14,318
- Murphy K (2007) Software for graphical models : A review. *ISBA (Intl Soc for Bayesian Analysis) Bulletin* 14(4):13–15

- Neapolitan R (1990) Probabilistic reasoning in expert systems: theory and algorithms. John Wiley and Sons, Inc. New York, NY, USA
- Nott D, Green P (2004) Bayesian variable selection and the swendsen- wang algorithm. *Journal of Computational and Graphical Statistics* 13:141–157
- Pearce D, Kelly P (2006) A dynamic topological sort algorithm for directed acyclic graphs. *ACM Journal of Experimental Algorithmics* 11:1–7
- Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann
- R Core Team (2013) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL <http://www.R-project.org/>
- Robert C, Casella G (2004) Monte Carlo Statistical Methods. Springer Texts in Statistics
- Robinson R (1973) Counting labeled acyclic digraphs. In: *New Directions in the Theory of Graphs*, New York Academic Press, pp 239–273
- Scutari M (2010) Learning bayesian networks with the bnlearn r package. *Journal of Statistical Software* 35:1–22